Representação e Processamento de Conhecimento na Web

Projeto Final

Repositório de Recursos Didáticos (RRD)

Luis Martins
PG47412

José Costa
PG47374

29 de junho de 2022

Conteúdo

Capítulo 1

Introdução

O projeto tem como objetivo desenvolver uma ferramenta Web que possa assistir no upload, manutenção e distribuição de recursos didáticos. Assim, esta aplicação possui características necessárias para o bom funcionamento de uma aplicação Web, autenticação de utilizadores, upload e download de recursos, funcionalidades restritas consoante o nível de acesso, possibilidade de comentar e avaliar os recursos disponíveis, criação e visualização de notícias, registo de logs (ações realizadas no sistema) e processamento dos mesmos. Para além destas funcionalidades, um utilizador com nível de acesso admin tem a possibilidade de gerir utilizadores, recursos e notícias.

Capítulo 2

Arquitetura aplicacional

A solução desenvolvida para o projeto baseia-se em micro-serviços, e para tal foram desenvolvidos os 3 servidores seguintes:

- O **API Server** que é responsável de responder a pedidos relativos aos recursos armazenados, bem como as noticias publicadas.
- O **Application Server** é responsável por responder ao pedidos feitos pelos vários clientes que utilizarão a aplicação, tudo isto através de uma interface na web. Este é o responsável por fazer a interligação entre os diferentes micro-serviços.
- O **Auth Server** é responsável pela autenticação dos utilizadores através da utilização de **JWT** que mantém a informação da sessão dos mesmos.

Para ter uma melhor ideia de como é que estes servidores comunicam entre si, foi desenvolvido um esquema de modo a facilitar a compreensão do mesmo.

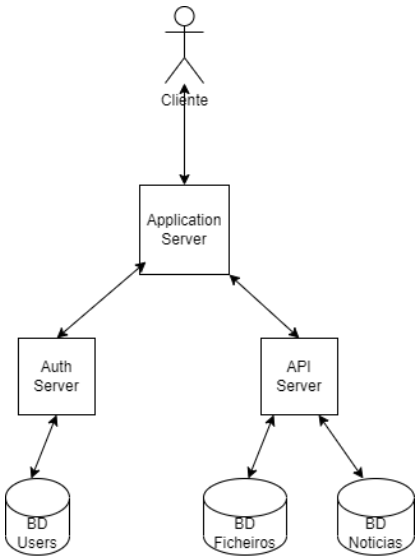

Figura 2.1: Arquitetura do sistema

Capítulo 3

Autorização de acesso

Para este projeto era pretendido que houve-se a implementação de diferentes tipos de utilizadores, cada um com as suas limitações, estando certas funcionalidades bloqueadas caso não se tenha nível de acesso grande o suficiente. No entanto as funcionalidades mais básicas (login, registar e ver as noticias na página inicial) não têm qualquer tipo de requerimento quanto ao nível de acesso, podendo ser utilizados por qualquer cliente que aceda à aplicação.

3.1 Consumidor

O **consumidor** é o nível mais baixo podendo apenas consultar os recursos e projetos que foram submetidos para a aplicação, bem como fazer download dos mesmos.

3.2 Produtor

Para além das funcionalidades que um consumidor tem acesso, um **Produtor** pode fazer upload de recursos no formato de um zip, que podem ser posteriormente **editados** pelo mesmo, podendo apenas editar os recursos que fez upload.

3.3 Administrador

Um administrador funciona como um produtor, no que toca ao fazer upload e editar recursos, mas este também tem outras funcionalidades, como:

- Editar, remover e adicionar utilizadores
- Editar qualquer recurso existente
- Adicionar, remover ou alterar noticias
- Aceder aos logs, bem como ter informação resumida sobre os mesmos

Capítulo 4

API Server

Este servidor foi desenvolvido com recurso a **Node.js** e à framework **Express**. Este fica à escuta, por *default*, na porta 8001, ou noutra porta caso tal seja especificado com uma variável de ambiente, e responde com metadados sobre os recursos e noticias armazenadas.

4.1 Modelos

De modo a permitir a persistências de dados, foi necessário criar, com auxilio do **mongoose**, dois schemas que representam os tipos de dados que irão ser armazenados. Estes schemas estarão armazenados numa Base de Dados em MongoDB de nome 'Projeto'. Um das collections da base de dados tem de nome **ficheiros**, onde estarão armazenados os metadados relativos aos diferentes ficheiros que serão uploaded e a outra collection tem de nome **noticias** e contem a informação relativa às noticias publicadas.

```
var ficheiroSchema = new mongoose.Schema({
    data_criacao: Number,
    data_submissao: String,
    id_prod: String,
    id_submissor: String,
    zip_name: String,
    nome_ficheiro: String,
    titulo_recurso: String,
    path_recurso: String,
    tipo_recurso: String,
    comentarios:[
      {
        "id_user": String,
        "data_criacao": String,
        "comentario": String
      }
    ],
    likedBy: [{type: String}],
    dislikedBy: [{type: String}]
  });
```

Figura 4.1: Schema do ficheiro

```
var noticiaSchema = new mongoose.Schema({
    data_criacao: String,
    titulo: String,
    conteudo: String,
    visibilidade: String
});
```

Figura 4.2: Schema da noticia

4.2 Controladores

De modo a permitir o acesso a cada uma das collections, foram criados 2 controladores, um para cada um dos schemas, recorrendo novamente ao módulo **mongoose**. Para os ficheiros implementamos as seguintes queries: listar, consultar, consultarUser, inserir, inserirComentario, remover, alterar, addLike, addDislike, removeLike e removeDislike.

Para as noticias temos as seguintes queries: listar, listarFiltered, inserir, remover, consultar, alterar, alterarVisibilidade e consultarUser.

4.3 Roteadores

De modo a facilitar a explicação das rotas todas que existem iremos apenas explicitar a parte da rota que se encontra depois do "/api", visto que todas as rotas tem isso em comum. De notar também que todas as rotas da API estão protegidas com recurso a um **JWT**, à exceção da rota GET para /noticias, pois decidimos permitir que pessoas não autenticadas possam ter acesso às notícias exibidas.

Método	Rota	Descrição
GET	/projetos	Lista todos os projetos que foram uploaded
GET	/recursos	Lista todos os recursos que foram uploaded
POST	/recursos	Inserção de um recurso
POST	/recursos/comentario/:id	Adiciona um comentário a um determinado recurso
POST	/recursos/like/:id	Adiciona um like de um user a um determinado recurso
POST	/recursos/dislike/:id	Adiciona um dislike de um user a um determinado recurso
DELETE	/recursos/like/:id	Remove um like de um user a um determinado recurso
DELETE	/recursos/dislike/:id	Remove um dislike de um user a um determinado recurso
GET	/recursos/user/:id	Lista todos os recursos que foram uploaded por um determinado user
GET	/recursos/:id	Lista a informação de um determinado recurso
PUT	/recursos/:id	Altera a informação de um determinado recurso
DELETE	/recursos/:id	Remove um determinado recurso da base de dados
GET	/noticias	Lista todas as noticias, podendo estas ser filtradas ou não por visibilidade
POST	/noticias	Inserção de uma noticia
DELETE	/noticias/:id	Remoção de uma noticia
GET	/noticias/:id	Lista informação de uma noticia
PUT	/recursos/:id	Altera a informação de uma determinada noticia

Capítulo 5

App Server

Este servidor, desenvolvido também com recurso a **Node.js** e à framework **Express**, é resposável pela criação das interfaces para humanos. Este servidor fica à escuta na porta default 8003 caso nenhuma porta seja especificada.

Esta camada é constituída por um roteador e várias templates das páginas para a ferramenta desenvolvida. É esta a camada que recebe os pedidos, e, para o satisfazer, faz pedidos à API de dados ou ao Servidor de Autenticação, recebe os dados e envia como resposta a renderização de uma template parametrizada com os dados recebidos. Estes templates estão escritos em PUG, usando ainda CSS, JavaScript, JQuery, AJAX. Usa ainda Chart.js para os gráficos da página de estatísticas.

Os pedidos feitos à API ou ao AuthServer são feitos para o localhost caso não estejam definidas variáveis de ambiente que indiquem qual é o domínio para onde os pedidos devem ser feitos.

5.1 Roteador

Todas as rotas definidas no roteador encontram-se na tabela em baixo. Algumas rotas estão protegidas através de níveis de acesso.

De modo a ser mais simples a criação de zips que obedecem à norma *bagit* foram criados 2 scripts identicos em Python.

O primeiro, que é usado por utilizadores, serve para comprimir uma pasta especifica, onde pode ser identificado o tipo de recursos bem como o id do produtor do recursos, bem como o nome do zip em questão.

Um segundo script, identico ao primeiro, é utilizado pelo App Server de modo a comprimir os ficheiros que foram descomprimidos, num novo zip no formato bagit, que será enviado ao utilizador que solicitou o download de um projeto.

Método	Rota	Descrição
GET	/	Página principal do projeto
GET	/recursos	Renderização da página que lista todos os recursos existentes
GET	/projetos	Renderização da página que lista todos os projetos existentes
GET	/projetos/:key	Inicia o download de um projeto
GET	/upload	Renderização da página principal de upload de projetos
POST	/upload	Envia um projeto para ser adicionado
GET	/download/:id	Inicia do download de um recurso
GET	/login	Renderização da página de login
POST	/login	Envio das credenciais para fazer login
GET	/registar	Renderização da página de registar
POST	/registar	Envio dos dados para registar utilizador
GET	/editar	Renderização da página de edição dos ficheiros
PUT	/editar/:id	Edição de um ficheiro
DELETE	/editar/:id	Remoção de um ficheiro
GET	/admin	Renderização da página inicial das ferramentas de admin
GET	/admin/utilizadores	Renderização da página de adição/edição/remoção de utilizadores
PUT	/admin/utilizadores/:id	Edição da informação de um utilizador
DELETE	/admin/utilizadores/:id	Remoção de um utilizador
POST	/admin/utilizadores/registar	Inserção de um utilizador
GET	/admin/recursos	Renderização da página de edição/remoção de ficheiros
PUT	/admin/recursos/:id	Edição da informação de um ficheiro
DELETE	/admin/recursos/:id	Remoção de um ficheiro
GET	/admin/estatisticas	Renderização da página de visualização de logs
GET	/admin/estatisticas/download	Inicia o download do ficheiro de logs
GET	/notícias	Retorna um json com notícias a começar numa certa notícia especificada
GET	/admin/noticias	Renderização da página de adição/edição/remoção de notícias
POST	/admin/noticias	Adição de uma notícia
DELETE	/admin/noticias/:id	Remoção de uma notícia
GET	/admin/noticia/:id	Retorna a informação de uma notícia
PUT	/admin/noticia	Altera a informação de uma notícia
POST	/comentario/:id	Inserção de um comentário
POST	/like/:id	Inserção de um like
POST	/deslike/:id	Inserção de um dislike
DELETE	/like/:id	Remoção de um like
DELETE	/dislike/:id	Remoção de um dislike
GET	/logout	Logout da aplicação

Capítulo 6

Auth Server

Este servidor foi desenvolvido com recurso a **Node.js** e à framework **Express**. Este fica à escuta, por *default*, na porta 8002, ou noutra porta caso tal seja especificado com uma variável de ambiente, e responde a pedidos relacionados com gestão de utilizadores e é responsável por criar **JSON Web Tokens** que são responsáveis por determinar os acessos a determinadas rotas nos servidores referidos anteriormente

6.1 Modelos

De modo a permitir a persistências de dados, foi necessário criar, com auxilio do **mongoose**, um schema que representa as informações sobre os utilizadores. Estes schemas estarão armazenados numa Base de Dados em MongoDB de nome 'Projeto'. Um das collections da base de dados tem de nome **users**, onde estarão armazenados os dados relativos aos diferentes utilizadores, como o seu username, password, que se encontra encriptada, e o nivel de acesso.

```
var userSchema = new mongoose.Schema({
    username: { type: String, required: true },
    password: { type: String, required: true },
    level: String
});
```

Figura 6.1: Schema do user

6.2 Controladores

De modo a permitir o acesso à collection dos users foi criado um controladores recorrendo novamente ao módulo **mongoose**. Para tal implementamos as seguintes queries: listar, consultar, inserir, remover e alterar.

6.3 Roteadores

Método	Rota	Descrição
GET	/	Lista todos os utilizadores
PUT	/	Altera a informação de um utilizador
DELETE	/	Remove um utilizador
POST	/login	Pedido de login, onde é feita a autenticação com a estratégia local, bem como a geração do **JWT**
POST	/registar	Adiciona um novo utilizador à base de dados, com a password encriptada

6.4 Estratégia de autenticação

Para todo o processo de autenticação utilizamos os módulos passport, passport-local e jsonwebtoken. Sempre que é feito um pedido de login é criado um novo token com o segredo "RPCWProjeto"e que ficará válido durante 1 hora. Do lado do app server, para se verificar a legitimidade do token utiliza-se o mesmo segredo. Para a estratégia local, ou seja, para verificar se a autenticação foi feita corretamente, acede-se à base de dados e verifica-se se as informações fornecidas são corretas, permitindo assim, posteriormente, criar um token válido.

Capítulo 7

Deployment

Para facilitar o deployment da aplicação foram criadas imagens docker, uma para cada um dos servidores, e estas mesmas foram colocadas no Docker Hub sob o nome zecosta109/api-server, zecosta109/app-server e zecosta109/auth-server. Para isto, foi utilizado um Dockerfile para cada um destes servidores.

De modo a ser mais fácil a atualização das imagens à medida que se fazia as alterações finais do projeto, foi criada uma Makefile que possibilita atualizar as imagens e fazer (re)upload das mesmas para os repositórios de Docker Hub mencionados anteriormente.

Nesta mesma Makefile existe uma opção para dar deploy dos containers utilizando docker-compose e o ficheiro docker-compose.yml, onde exportamos variáveis de ambiente, de modo a poder distinguir se estávamos a lançar a aplicação diretamente no localhost ou a partir dos containers. Neste docker-compose também definimos os persistent volumes para os recursos que foram uploaded, para a base de dados e para os ficheiros de log, de modo a ter persistência de dados entre diferentes execuções.

Capítulo 8

Resultados

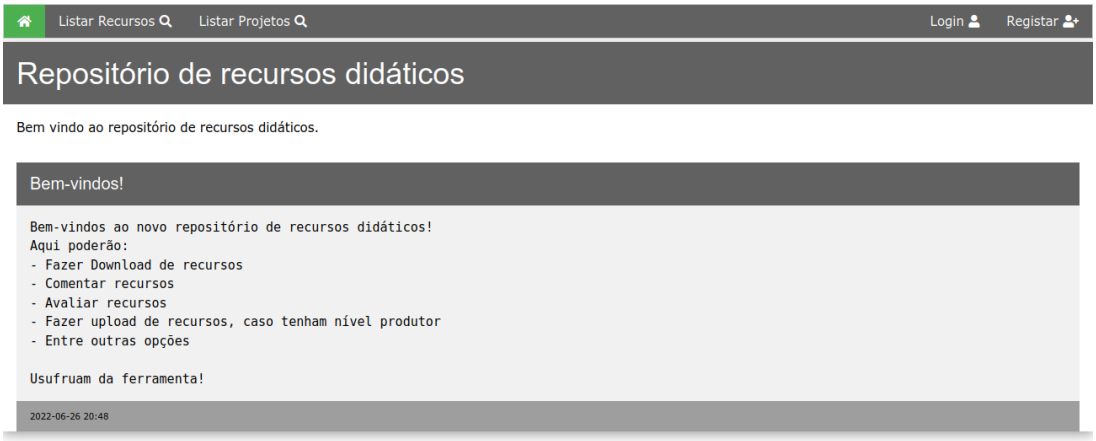

Figura 8.1: Página Principal

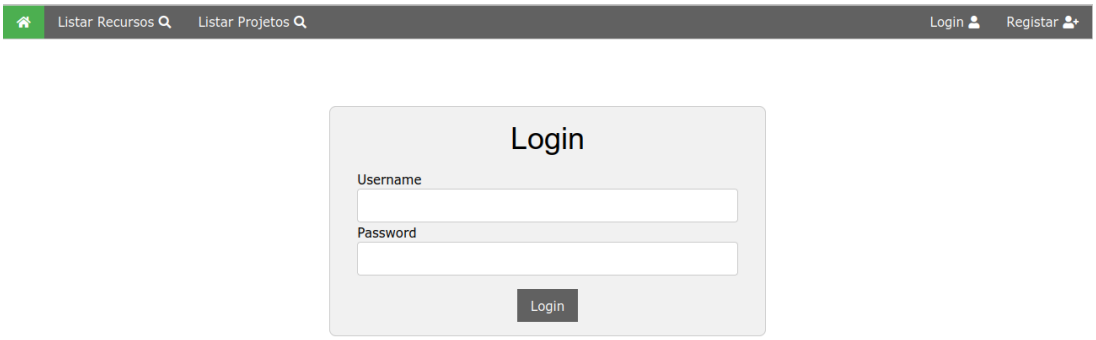

Figura 8.2: Página de Login

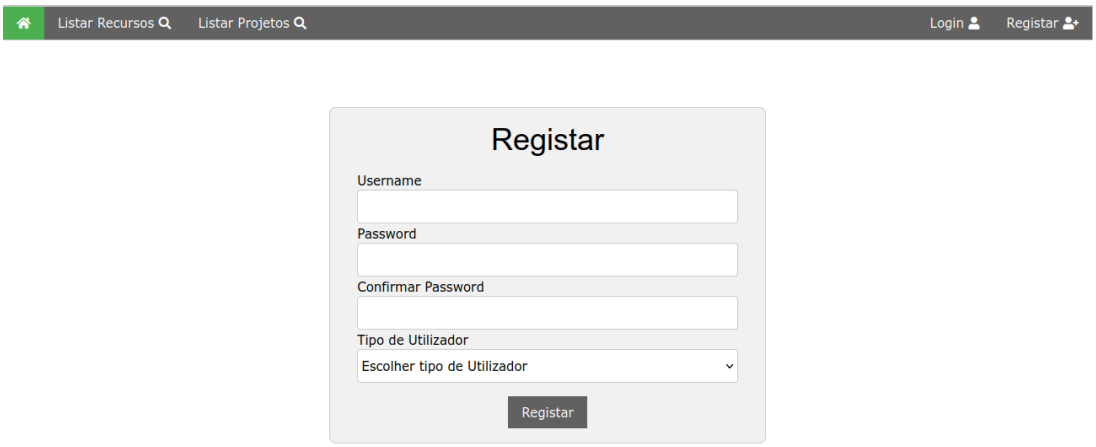

Figura 8.3: Página de Register

Figura 8.4: Página de Uploads

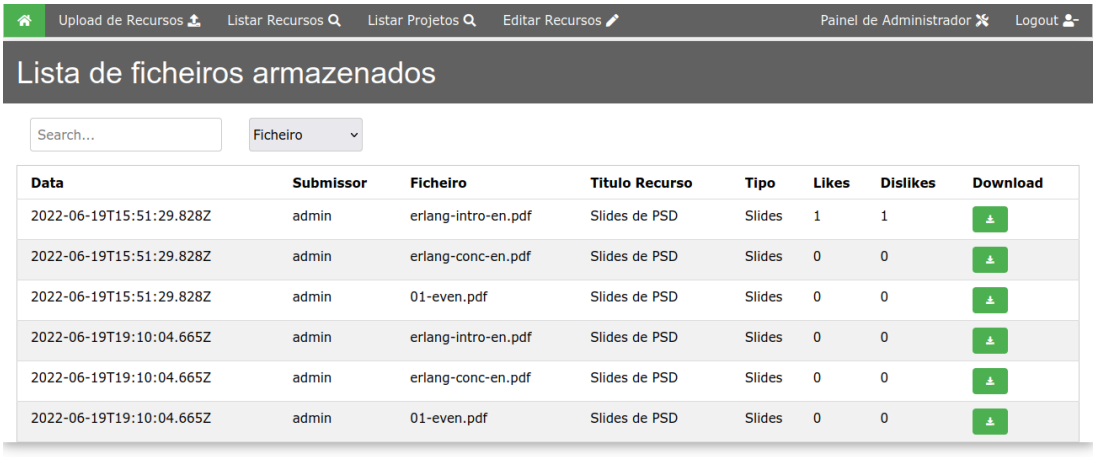

Figura 8.5: Página de Listar Recursos

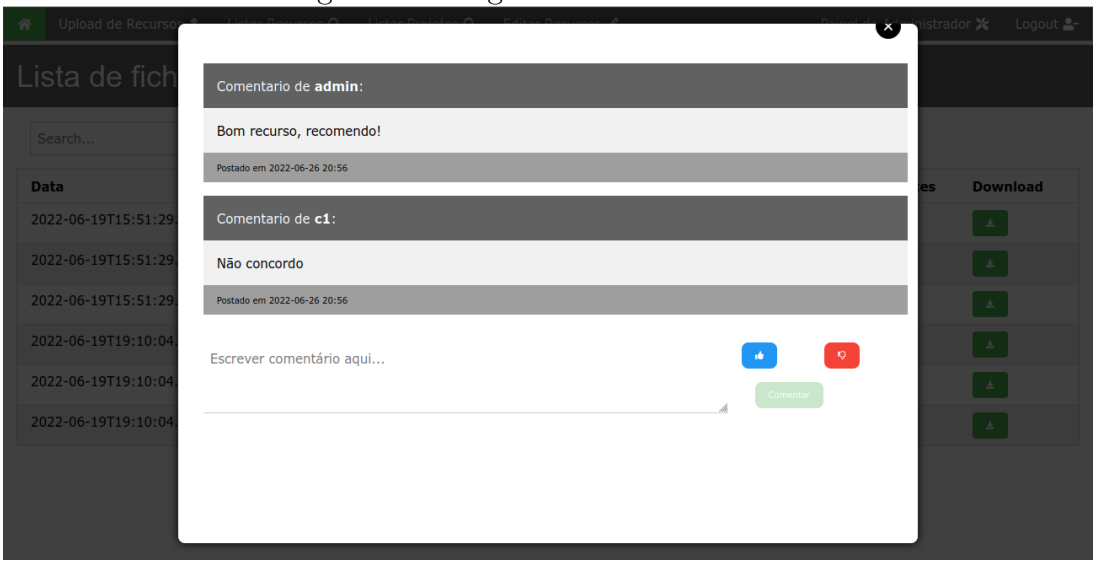

Figura 8.6: Página de Listar Recursos - Comentários

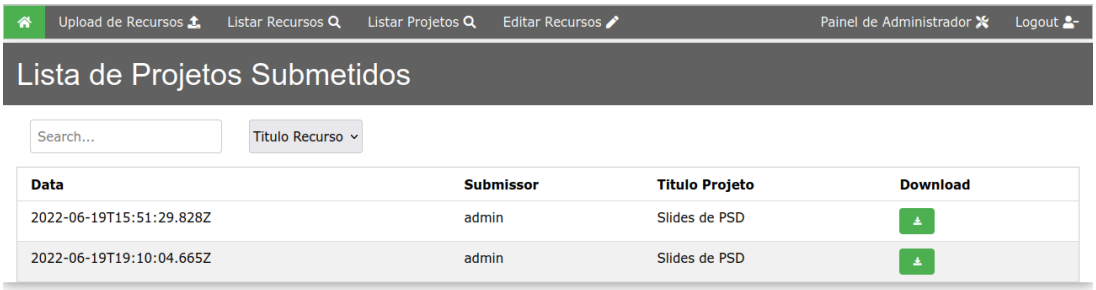

Figura 8.7: Página de Listar Projetos

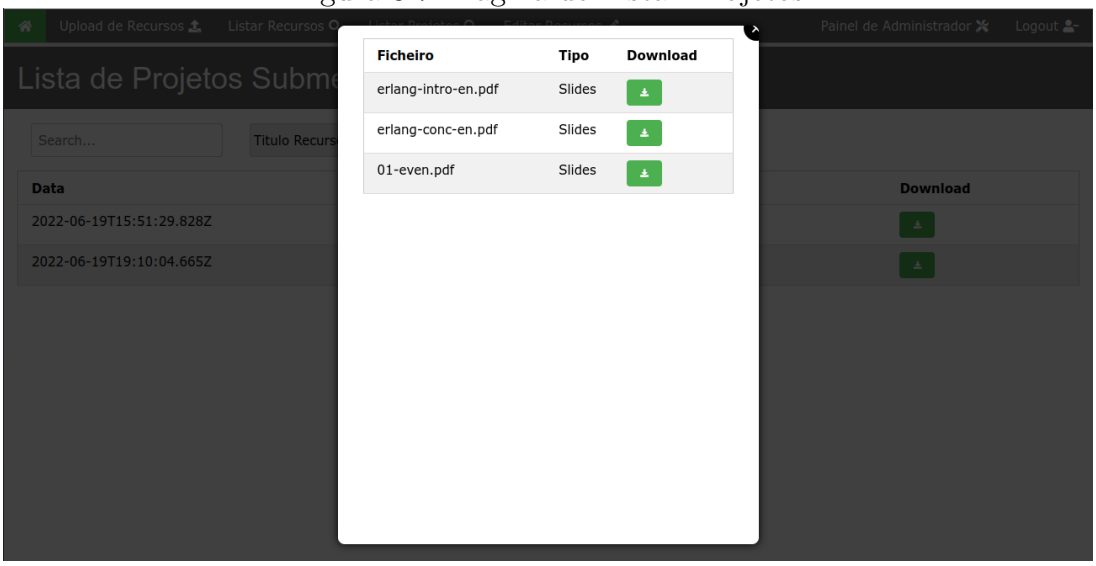

Figura 8.8: Página de Listar Projetos - Lista de Recursos

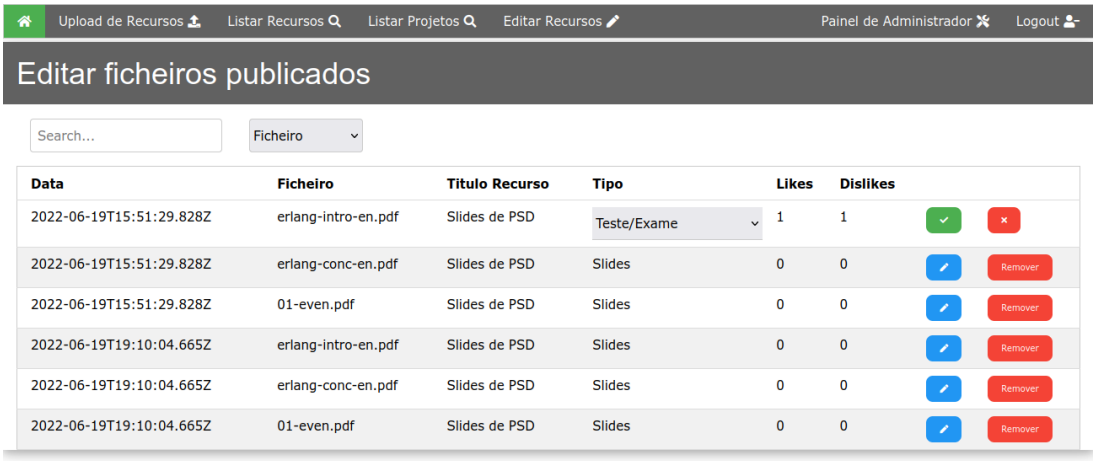

Figura 8.9: Página de Editar Recursos

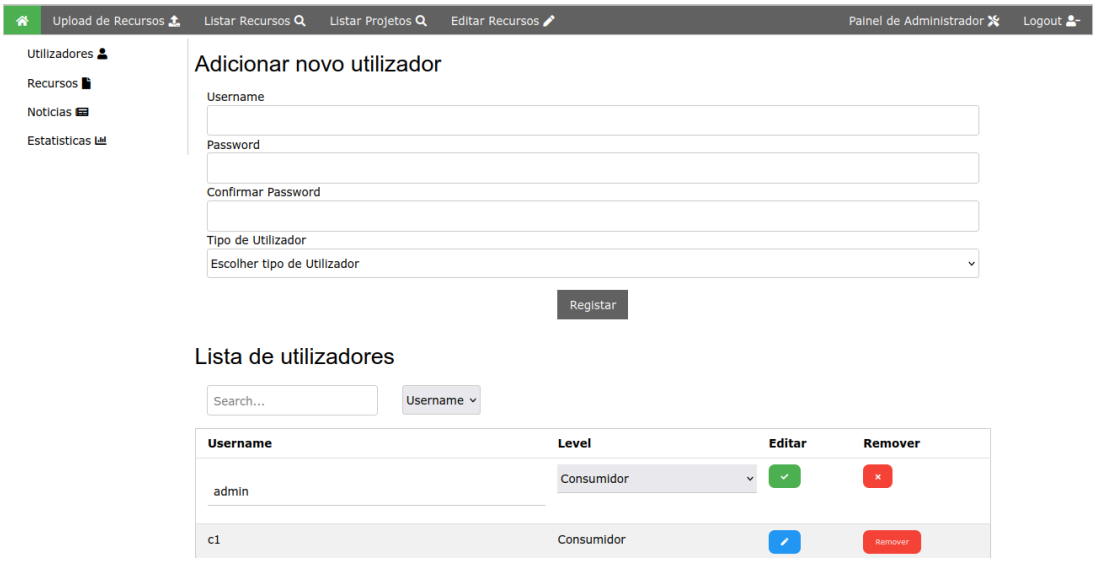

Figura 8.10: Página de Administrador - Utilizadores

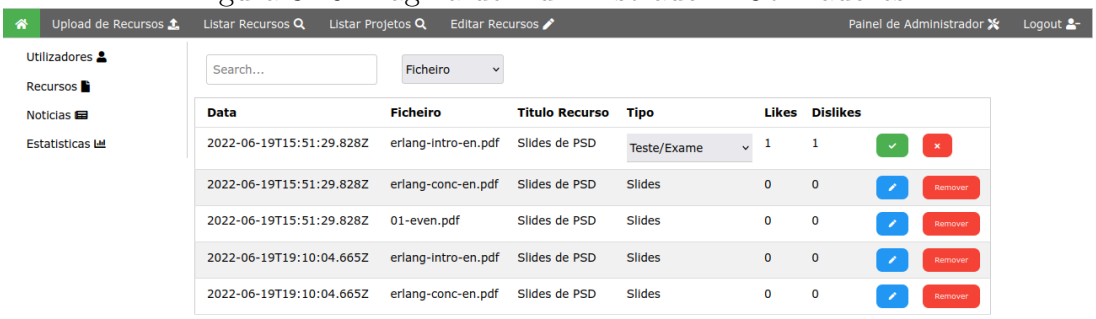

Figura 8.11: Página de Administrador - Recursos

Figura 8.12: Página de Administrador - Noticias

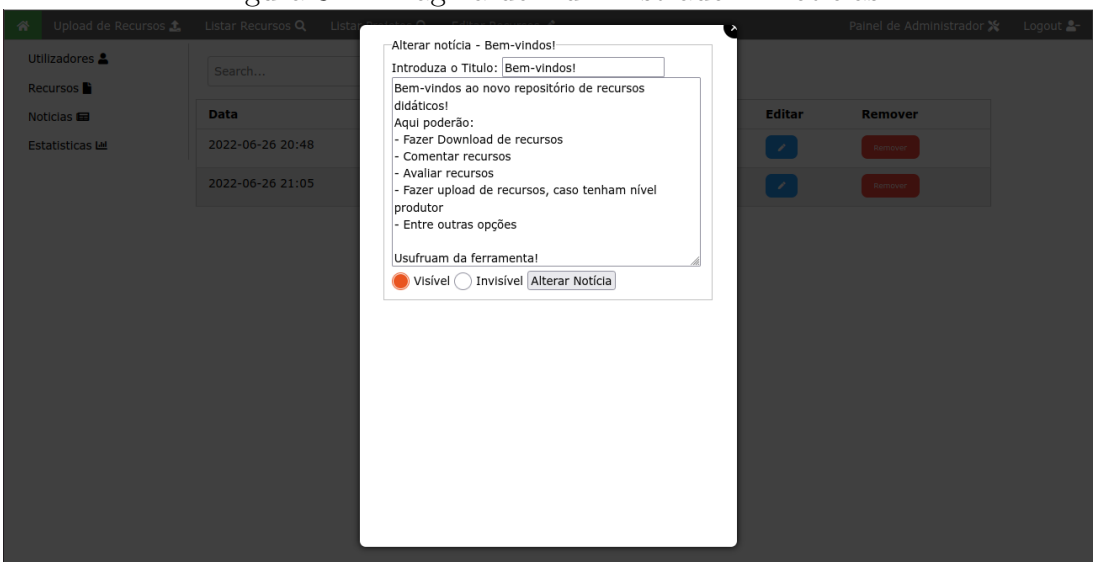

Figura 8.13: Página de Administrador - Noticias com Edição

Figura 8.14: Página de Administrador - Estatísticas

```
 72 upload|upload|2022-06-26T21:11|p1
 73 upload|upload|2022-06-26T21:11|p1
 74 upload|upload|2022-06-26T21:11|p1
 75 upload|upload|2022-06-26T21:11|p1
 76 login|succ|2022-06-26T21:11|admin
 77 login|succ|2022-06-26T21:12|p1
 78 recursos|vis|2022-06-26T21:12|p1
 79 projetos|vis|2022-06-26T21:12|p1
 80 recursos|vis|2022-06-26T21:12|p1
 81 projetos|vis|2022-06-26T21:12|p1
 82 recursos|vis|2022-06-26T21:12|p1
 83 projetos|vis|2022-06-26T21:12|p1
 84 projetos|vis|2022-06-26T21:12|p1
 85 recursos|edit|2022-06-26T21:12|p1|62b8cb6ea86bf0001a3914a0
 86 projetos|vis|2022-06-26T21:12|p1
 87 projetos|down|2022-06-26T21:12|p1|ed831955cfc948dd9234bcf005d9f097
 88 projetos|down|2022-06-26T21:12|p1|ed831955cfc948dd9234bcf005d9f097
 89 projetos|down|2022-06-26T21:12|p1|925f9ead3c72afb288a0351807484576
 90 projetos|down|2022-06-26T21:12|p1|aab0c4364a252d3396295323a2e02bc9
 91 recursos|vis|2022-06-26T21:12|p1
 92 recursos|down|2022-06-26T21:12|p1|erlang-conc-en.pdf
 93 recursos|down|2022-06-26T21:12|p1|erlang-func-en.pdf
 94 recursos|down|2022-06-26T21:12|p1|erlang-intro-en.pdf
 95 recursos|down|2022-06-26T21:13|p1|01-even.pdf
 96 login|succ|2022-06-26T21:13|admin
 97 registar|succ|2022-06-26T21:13|p2|Produtor
 98 login|succ|2022-06-26T21:13|p2
 99 upload|upload|2022-06-26T21:14|p2
100 upload|upload|2022-06-26T21:14|p2
101 upload|upload|2022-06-26T21:14|p2
102 projetos|vis|2022-06-26T21:14|p2
103 recursos|vis|2022-06-26T21:14|p2
```

Figura 8.15: Exemplo do ficheiro logs

Capítulo 9

Conclusão

Com este projeto foi possível aprofundar os conceitos abordados na disciplina, como o desenvolvimento de aplicações web utilizando micro-serviços que comunicam entre si, autenticação de utilizadores através de Passport e tokens JWT, a base de dados nosql orientada a documentos MongoDB, interfaces gráficas dinâmicas e *deployment* através de docker; e criar uma aplicação final capaz de fornecer um serviço útil para os clientes.

Foi também possível aprender novos conceitos, como processos de ingestão de recursos para arquivo e disseminação dos mesmos.

Tendo cumprido tudo o que foi pedido no inicio do desenvolvimento do projeto, conseguimos notar que este poderia ainda ser estendido, implementando novas funcionalidades.

De modo geral, estamos muito satisfeitos com o resultado final e consideremos que poderá vir a ter utilidade no futuro.